CA 1

1.a Closed form solution of Ridge Regression

We need to minimise

$Min_w \ \left(\frac{1}{N}\sum_i ||x_i^T w - y_i||^2 + \lambda||w||^2\right)$
$= Min_w \ \left(\sum_i ||x_i^T w - y_i||^2 + \alpha||w||^2\right)$, where $\alpha = N\lambda$.

$w^* = ArgMin_w \ \left(||w^T X - Y||^2 + \alpha||w||^2\right)$, where
$X = [x_1 \ x_2 \ \ldots], Y = [y_1 \ y_2 \ \ldots]$.

Taking derivative w.r.t w and equating it to zero gives

$2X(w^\top X - Y)^T + 2\alpha w = 0$
$\Rightarrow XX^T w - XY^T + \alpha w I = 0$
$\Rightarrow w^* = \left(XX^T + \alpha I\right)^{-1} XY^T$

Group 3: Nice!

If we assume an error variance of σ_1^2 and a prior variance σ_2^2, then
$N\lambda = \alpha = \sigma_1^2/\sigma_2^2$.

1.b Household power dataset

In [ ]:

```python
import pandas as pd
import numpy as np
import time as tm

giturl = 'https://raw.githubusercontent.com/vnmo/MLon/main/CA1_1b_housel
data  = pd.read_csv(giturl)

# This is a cleaned data using MATLAB for easiness.
# Date and Time are combined and converted to UNIX format
##      --- one can subtract any date base from this if required to red
##      --- For example, lowest date time is 21/12/2006 11:17:00. Subtr
##      --- Subtract  1164931200 to make the base to 01-Dec-2006 00:00:
##      --- Subtract  1136073600 to make the base to 01-Jan-2006 00:00:
# All NaNs and invalid cells are deleted.
# Hence new size is 2049280 x 8
#-----Attributes-----
#           Date Time in UNIX format
#           Active Power
#           Reactive Power
#           Voltage
#           Global Intensity
#           Submetering 1
#           Submetering 2
#           Submetering 3
```

Group 3: This can be removed

In [ ]:

```python
# Take  Powers, voltage and intensity as input variable, i.e., x_i for .
inputdata= data.iloc[:, [1, 2, 3, 4]]

# Take all sub-meterings as output variable, i.e., y_i for i=1:N
# Change if required
outputdata=data.iloc[:,[5,6,7]]

X=np.transpose(inputdata)
Y=np.transpose(outputdata)
print(f"Size of input data = {X.shape}")
print(f"Size of output data = {Y.shape}")

alpha = np.float32(2.0) # Lambda: Take a value
p = len(X) # Dimension of data

start = tm.time()
# Find the regressor matrix (or vector) using the closed-form solution
w_opt = np.linalg.inv((X@X.T) + alpha * np.eye(p)) @ (X@Y.T)
stop = tm.time()

print(f"Regressor w=\n {w_opt}")
totTime_1b = stop-start
print(f"\nTotal time time taken to compute the closed form solution = {
```

Group 3: Maybe not needed?

Group 3: Can be removed

```
Size of input data = (4, 2049280)
Size of output data = (3, 2049280)
Regressor w=
    Sub_metering_1  Sub_metering_2  Sub_metering_3
0      -14.082426      -14.391115       48.262268
1       -3.114029       -1.534777        3.020558
2       -0.007566       -0.005523        0.004318
3        4.041872        4.004468      -10.296116
```

Group 3: Nice!

```
Total time time taken to compute the closed form solution = 1.1671276092
529297 seconds
```

Group 3: Interesting but maybe not needed?

1.c Greenhouse dataset

In [ ]:

```
import pandas as pd
import numpy as np
import time as tm

giturl1 = 'https://raw.githubusercontent.com/vnmo/MLon/main/CA1_1c_greer
giturl2 = 'https://raw.githubusercontent.com/vnmo/MLon/main/CA1_1c_greer
data1  = pd.read_csv(giturl1)
data2  = pd.read_csv(giturl2)
data = pd.concat([data1,data2],ignore_index=True)

# This is a cleaned data using MATLAB for easiness.
# Data is split into 4 parts due to github data restrictions
```

In [ ]:

```
# Input variables (Change if required)
inputAttributes = np.arange(0,5231)
# Output variables (Change if required)
outputAttributes = np.arange(5231,5232)

inputdata=data.iloc[:,inputAttributes]
outputdata=data.iloc[:,outputAttributes]
X=np.transpose(inputdata)
Y=np.transpose(outputdata)
print(f"Size of input data = {X.shape}")
print(f"Size of output data = {Y.shape}")

alpha = np.float32(2.0) # Lambda: Take a value
p = len(X) # Dimension of data

start = tm.time()
# Find the regressor matrix (or vector) using the closed-form solution
w_opt = np.linalg.inv((X@X.T) + alpha * np.eye(p)) @ (X@Y.T)
stop = tm.time()

print(f"Regressor w=\n {w_opt}")
totTime_1c = stop-start
print(f"\nTotal time time taken to compute the closed form solution = {
```

Group 3: Same as on the last page

```
Size of input data = (5231, 2921)
Size of output data = (1, 2921)
Regressor w=
            5231
0    -0.102481
1    -0.142095
2     0.751174
3    -0.521801
4     0.258978
...        ...
5226 -1.081724
5227  0.025382
5228 -1.059141
5229  1.886304
5230  0.132544

[5231 rows x 1 columns]

Total time time taken to compute the closed form solution = 25.548195362
091064 seconds
```

Group 3: Nice!

There is scalability issue and the time taken for second dataset is around 20 to 1000 times more than that of first dataset.

1.d

As is evident from $\Rightarrow w^* = \left(XX^T + \alpha I\right)^{-1} XY^T$ calculated above, this least sqaures problem becomes harder to compute as the matrices increase in size and become denser. Therefore, while linear regression offers a one step solution, because of being computationally expensive, iterative methods such as gradient descent need to be employed.

Group 3: Sorry, but this paragraph is a little hard to read...

Group 3: Calculated how? Would be nice if you could show the big-o-notation! :)

Group 3: I'd rephrase this to: "This makes iterative methods like gradient descent a more suitable option than linear regression as the matrices become larger and/or denser.".

Group 3: Would be nice if you could show an equation to demonstrate how the computational load per iteration is lower in gradient descent.